On the Exploration of the Query Rewriting Space with Existential Rules

Mélanie König, Michel Leclère, Marie-Laure Mugnier, Michaël Thomazo

On the Exploration of the Query Rewriting Space with Existential Rules

Mélanie König, Michel Leclère, Marie-Laure Mugnier, and Michaël Thomazo

University Montpellier 2, France

Abstract. We address the issue of Ontology-Based Data Access, with ontologies represented in the framework of existential rules, also known as Datalog+/-. A well-known approach involves rewriting the query using ontological knowledge. We focus here on the basic rewriting technique which consists of rewriting a conjunctive query (CQ) into a union of CQs. We assume that the set of rules is a finite unification set, i.e., for any CQ, there exists a finite sound and complete rewriting of this CQ with the rules. First, we study a generic breadth-first rewriting algorithm, which takes as input any rewriting operator. We define properties of the rewriting operator that ensure the correctness and the termination of this algorithm. Second, we study some operators with respect to the exhibited properties. All these operators have in common to be based on so-called piece-unifiers but they lead to different explorations of the rewriting space. Finally, an experimental comparison of these operators within an implementation of the generic breadth-first rewriting algorithm is presented.

1 Introduction

We address the issue of Ontology-Based Data Access, which aims at exploiting knowledge expressed in ontologies while querying data. In this paper, ontologies are represented in the framework of existential rules [BLMS11,KR11], also known as Datalog$\pm$ [CGK08,CGL09]. Existential rules allow to assert the existence of new unknown individuals, which is a crucial feature in an open-world perspective, where data are incompletely represented. These rules are of the form $body \rightarrow head$, where the body and the head are conjunctions of atoms (without functions) and variables that occur only in the head are *existentially* quantified. They generalize lightweight description logics, which form the core of the tractable profiles of OWL2.

The general query answering problem can be expressed as follows: given a knowledge base $\mathcal{K}$ composed of data and an ontology (a set of existential rules here), and a query Q, compute the set of answers to Q in $\mathcal{K}$. In this paper, we consider Boolean conjunctive queries (note however that all our results are easily extended to non-Boolean conjunctive queries). The fundamental question becomes: is Q entailed by $\mathcal{K}$?

There are two main approaches to solve this problem, which are linked to the classical paradigms for processing rules, namely forward and backward chaining. Both can be seen as ways of reducing the problem to a classical database query answering problem by eliminating the rules. The first approach consists of applying the rules to the data, thus materializing entailed facts into the data. Then, Q is entailed by $\mathcal{K}$ if and only if it can be mapped to this materialized database. The second approach consists of

using the rules to rewrite the query into a first-order query (typically a union of conjunctive queries [CGL+07,PUHM09,GOP11,VSS12,RMC12]) or a Datalog query [RA10]. Then, Q is entailed by $\mathcal{K}$ if and only if the rewritten query is entailed by the initial database. Finally, techniques combining both approaches are developed, in particular the so-called combined approach [LTW09,KLT+11].

In this paper, we focus on rewriting techniques, and more specifically on rewriting the initial conjunctive query Q into a union of conjunctive queries, that we will see as a set of conjunctive queries, called *rewritings* of Q. The goal is to compute a set of rewritings both *sound* (if one of its elements maps to the initial database, then $\mathcal{K}$ entails Q) and *complete* (if $\mathcal{K}$ entails Q then there is an element that maps to the initial database). Minimality may also be a desirable property.

As in classical backward chaining, the rewriting process relies on a unification operation between the current query and a rule head. However, existential variables in rule heads induce a structure that has to be considered to keep soundness. Thus, instead of unifying a single atom of the query at once, our unifier processes a subset of atoms from the query. A *piece* is a minimal subset of atoms from the query that have to be erased together, hence the name *piece-unifier*. Piece-unifiers lead to a logically sound and complete rewriting method. As far as we know, it is the only method accepting any kind of existential rules, while staying in this fragment, i.e., without Skolemization of rule heads to replace existential variables with Skolem functions.

Computing a set of rewritings can be reformulated in terms of exploring a potentially infinite space of queries, composed of the initial (Boolean) conjunctive query and its sound rewritings, with the aim of computing a complete set of rewritings. This space can be provided with a partial preorder, such that $Q_2 \geq Q_1$ (Q_2 is more general than Q_1) if there is a homomorphism from Q_2 to Q_1. It can be shown that the completeness of the output set is kept when this set is restricted to its most general elements.

We recall that the entailment problem with existential rules is undecidable. A set of existential rules ensuring that a finite sound and complete set of most general rewritings exists for any query is called a *finite unification set* (fus) [BLMS11]. Note that, in the case of fus rules, it may be the case that the set of sound rewritings of the query is infinite while the set of its most general sound rewritings is finite. It follows that a breadth-first exploration of the rewriting space is not sufficient to ensure finiteness of the process; one also has to maintain a set of the most general rewritings. At each step of the breadth-first algorithm, some queries are thus discarded, because they are more specific than another rewriting, even if they have not been explored yet. The question is whether this dynamic pruning of the search space guarantees the completeness of the output. This is the main point at the origin of this paper. This ties in with an issue raised in [ISG12] about the gap between theoretical completeness of some methods and the effective completeness of their implementation, this gap being mainly due to errors in algorithmic optimizations.

Paper contributions. The global breadth-first algorithm maintains a set of rewritings $\mathcal{Q}$ and iteratively performs the following task until all queries from $\mathcal{Q}$ have been explored: (1) generate all one-step rewritings from unexplored queries in $\mathcal{Q}$; (2) add these rewritings to $\mathcal{Q}$ and keep only the most general elements in $\mathcal{Q}$. We call *rewriting operator* the function that, given a query and a set of rules, returns a set of one-step rewritings of

this query with the rules. The question raised can be expressed as follows: under what conditions a rewriting operator proven to be complete leads to a complete set when the space of rewritten queries is pruned at each step of the breadth-first algorithm? More generally, which properties have to be fulfilled by the operator to ensure correctness and termination of the algorithm? To answer this question, we define several properties that a rewriting operator has to satisfy and show that they actually ensure correctness and termination of the algorithm: soundness, completeness, prunability and finite coverability.

We then study several operators based on piece-unification in light of these properties. We point out that it follows from the results in [KLMT12] that the piece-based rewriting operator is sound, complete and prunable. These properties still hold when only most general piece-unifiers are considered. The picture is not the same when we consider the restriction to piece-unifiers processing a single piece at once; whereas the single-piece based operator is sound and complete, as proven in [KLMT12], it is not prunable. We exhibit several examples for which the output is not a complete set of rewritings. Thus, if single-piece unifiers are interesting from an algorithmic viewpoint, they have to be combined to achieve prunability. We then introduce a new piece-based rewriting operator, called an aggregator, which explores the space of rewritings in a radically different way. This operator is shown to be sound, complete and prunable. However, for this operator to become more efficient than the previous ones, we provided it with an optimization. According to experiments the new operator generates significantly less queries than the other piece-based operators and outputs a complete rewriting set. However the prunability of the optimized operator is not proven theoretically yet.

The paper is organized as follows. Section 2 recalls some preliminaries on the existential rule framework. In Section 3 the generic breadth-first algorithm is introduced and general properties of rewriting operators are studied. In Section 4, we focus on piece-based unifiers, and their restrictions to most general piece-unifiers and single-piece unifiers. Section 5 is devoted to the new agregation operator. Section 6 presents ongoing work on optimization, experiments and draws some perspectives. The proofs of the results are available in the accompanying report [KLMT13].

2 Framework

2.1 Preliminaries

An *atom* is of the form $p(t_1, \ldots, t_k)$ where p is a predicate with arity k, and the t_i are terms, i.e., variables or constants. Given an atom or a set of atoms A, *vars*(A), *consts*(A) and *terms*(A) denote its set of variables, of constants and of terms, respectively. In the following examples, all the terms are variables (denoted by x, y, z, etc.). $\models$ denotes the classical logical consequence.

A *fact* is an existentially closed conjunction of atoms.[1] A *conjunctive query* (CQ) is an existentially quantified conjunction of atoms. When it is a closed formula, it is called a *Boolean* CQ (BCQ). Hence facts and BCQs have the same logical form. In

[1] We generalize the classical notion of a fact in order to take existential variables into account.

the following, we will see them as sets of atoms. Given sets of atoms A and B, a *homomorphism* h from A to B is a substitution of *vars*(A) by *terms*(B) s.t. $h(A) \subseteq B$. We say that A is *mapped* to B by h. If there is a homomorphism from A to B, we say that A is *more general* than B, which is denoted $A \geq B$. Given a fact F and a BCQ Q, the answer to Q in F is *positive* if $F \models Q$. It is well-known that $F \models Q$ iff there is a homomorphism from Q to F.

Definition 1 (Existential rule). *An* existential rule *(or simply a* rule*) is a formula* $R = \forall \mathbf{x} \forall \mathbf{y}(B[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{z} H[\mathbf{y}, \mathbf{z}])$ *where* $B =$ *body*(R) *and* $H =$ *head*(R) *are conjunctions of atoms, resp. called the* body *and the* head *of* R*. The* frontier *of* R*, noted* fr(R)*, is the set* vars$(B) \cap$ vars$(H) = \mathbf{y}$*. The set of* existential variables *in* R *is the set* vars$(H) \setminus$ *fr*$(R) = \mathbf{z}$*.*

In the following, we will omit quantifiers in rules as there is no ambiguity.

A *knowledge base* (KB) $\mathcal{K} = (F, \mathcal{R})$ is composed of a fact F and a finite set of existential rules $\mathcal{R}$. The *BCQ entailment problem* takes as input a KB $\mathcal{K} = (F, \mathcal{R})$ and a BCQ Q, and asks if $F, \mathcal{R} \models Q$ holds.

2.2 Desirable Properties of Rewriting Sets

Given a query Q and a set of existential rules $\mathcal{R}$, rewriting techniques compute a set of queries $\mathcal{Q}$, called a *rewriting set*. It is generally desired that such a set satisfies at least three properties: *soundness*, *completeness* and *minimality*.

Definition 2 (Sound and Complete (rewriting) set of BCQs). *Let* $\mathcal{R}$ *be a set of existential rules and* Q *be a BCQ. Let* $\mathcal{Q}$ *be a set of CQs.* $\mathcal{Q}$ *is said to be* sound *w.r.t.* Q *and* $\mathcal{R}$ *if for all facts* F*, for all* $Q' \in \mathcal{Q}$*, if* Q' *can be mapped to* F *then* $\mathcal{R}, F \models Q$*. Reciprocally,* $\mathcal{Q}$ *is said to be* complete *w.r.t.* Q *and* $\mathcal{R}$ *if for all fact* F*, if* $\mathcal{R}, F \models Q$ *then there is* $Q' \in \mathcal{Q}$ *s.t.* Q' *can be mapped to* F*.*

To define the minimality notion, we use the following covering relation among sets of BCQs.

Definition 3 (Covering relation). *Let* $\mathcal{Q}_1, \mathcal{Q}_2$ *be two sets of BCQs.* $\mathcal{Q}_1$ covers $\mathcal{Q}_2$*, which is denoted* $\mathcal{Q}_1 \geq \mathcal{Q}_2$*, if for each* $Q_2 \in \mathcal{Q}_2$ *there exists* $Q_1 \in \mathcal{Q}_1$ *s.t.* $Q_1 \geq Q_2$*.*

Definition 4 (Minimal set of BCQs). *Let* $\mathcal{Q}$ *be a set of BCQs.* $\mathcal{Q}$ *is said to be* minimal *if there is no* $Q \in \mathcal{Q}$ *such that* $(\mathcal{Q} \setminus \{Q\}) \geq \mathcal{Q}$*.*

In [KLMT12] it is shown that, given a finite set of existential rules $\mathcal{R}$ and a BCQ Q, all sound, complete and minimal rewritings sets have the same cardinality. Furthermore, any sound and complete finite rewriting set can be made minimal by selecting one of its minimal covering subsets, i.e., $\mathcal{Q}' \subseteq \mathcal{Q}$ s.t. $\mathcal{Q}'$ is minimal and $\mathcal{Q}' \geq \mathcal{Q}$.

3 A Generic Breadth-First Rewriting Algorithm

We will now present a generic rewriting algorithm that takes a set of existential rules and a query as input and a *rewriting operator* as parameter. The studied question is the following: which properties should this operator fulfill in order that the algorithm outputs a sound, complete and minimal set?

3.1 Algorithm

Definition 5 (Rewriting operator). *A* rewriting operator $\texttt{rew}$ *is a function which takes as input a conjunctive query Q and a set of rules $\mathcal{R}$ and outputs a set of conjunctive queries $\texttt{rew}(Q, \mathcal{R})$.*

Since the elements of $\texttt{rew}(Q, \mathcal{R})$ are queries, it is possible to apply further steps of rewriting to them. This naturally leads to the notions of k-rewriting and k-saturation.

Definition 6 (k-rewriting). *Let Q be a conjunctive query, $\mathcal{R}$ be a set of rules and $\texttt{rew}$ be a rewriting operator. A 1-rewriting of Q (w.r.t. $\texttt{rew}$ and $\mathcal{R}$) is an element of $\texttt{rew}(Q, \mathcal{R})$. A k-rewriting of Q, for $k > 1$, (w.r.t. $\texttt{rew}$ and $\mathcal{R}$) is a 1-rewriting of a $(k-1)$-rewriting of Q.*

Definition 7 (k-saturation). *Let Q be a query, $\mathcal{R}$ be a set of rules and $\texttt{rew}$ be a rewriting operator. We denote by $\texttt{rew}_k(Q, \mathcal{R})$ the set of k-rewritings of Q. We call k-saturation, and denote by $W_k(Q, \mathcal{R})$, the set of i-rewritings of Q for any $i \leq k$. We denote $W_\infty(Q, \mathcal{R}) = \bigcup_{k \in \mathbb{N}} W_k(Q, \mathcal{R})$.*

In the following, we extend the notations $\texttt{rew}$, $\texttt{rew}_k$ and W_k to a set of queries $\mathcal{Q}$ instead of a single query Q: $\texttt{rew}(\mathcal{Q}, \mathcal{R}) = \bigcup_{Q \in \mathcal{Q}} \texttt{rew}(Q, \mathcal{R})$, $\texttt{rew}_k(\mathcal{Q}, \mathcal{R}) = \bigcup_{Q \in \mathcal{Q}} \texttt{rew}_k(Q, \mathcal{R})$ and $W_k(\mathcal{Q}, \mathcal{R}) = \bigcup_{i \leq k} \texttt{rew}_i(\mathcal{Q}, \mathcal{R})$.

Algorithm 1 performs a breadth-first exploration of the rewriting space of a given query. At each step, only the most general elements are kept thanks to a covering function, denoted by $\texttt{cover}$, that computes a minimal covering subset of a given set.

If $\texttt{rew}$ fulfills some good properties (subsequently specified), then after the i^{th} iteration of the while loop the i-saturation of Q (with respect to $\mathcal{R}$ and $\texttt{rew}$) is covered by $\mathcal{Q}_F$, while $\mathcal{Q}_E$ contains the queries that remain to be explored.

Algorithm 1: A GENERIC BREADTH-FIRST REWRITING ALGORITHM

Data: A fus $\mathcal{R}$, a conjunctive query Q
Access : A rewriting operator $\texttt{rew}$, a minimal covering function $\texttt{cover}$
Result: A minimal cover of the set of all the rewritings of Q
$\mathcal{Q}_F \leftarrow \{Q\}$; *// resulting set*
$\mathcal{Q}_E \leftarrow \{Q\}$; *// queries to be explored*
while $\mathcal{Q}_E \neq \emptyset$ **do**
  $\mathcal{Q}_C \leftarrow \texttt{cover}(\mathcal{Q}_F \cup \texttt{rew}(\mathcal{Q}_E, \mathcal{R}))$; *// update cover*
  $\mathcal{Q}_E \leftarrow \mathcal{Q}_C \backslash \mathcal{Q}_F$; *// select unexplored queries*
  $\mathcal{Q}_F \leftarrow \mathcal{Q}_C$;
return $\mathcal{Q}_F$

In the following, we study the conditions that a rewriting operator must meet in order that: (i) the output set is a minimal cover of the set of all the rewritings that can be obtained by using this rewriting operator on the inputs, (ii) the output set is sound and complete, according to Definition 2, and (iii) the algorithm halts. We introduce the notion of *prunable operator* for the first condition, which will be combined with that of

sound and complete operator for the second one. Finally a notion of *finitely coverable operator* is introduced for the third one.

3.2 Correctness of the algorithm

We now exhibit a sufficient property on the rewriting operator that ensures that Algorithm 1 outputs a minimal cover of $W_\infty(Q, \mathcal{R})$.

Definition 8 (Prunable). *Let $\mathcal{R}$ be a set of rules and $\texttt{rew}$ be a rewriting operator. $\texttt{rew}$ is* prunable *if for all queries Q_1, Q_2, Q'_2 such that $Q_1 \geq Q_2$, $Q'_2 \in \texttt{rew}(Q_2, \mathcal{R})$ and $Q_1 \not\geq Q'_2$, there exists $Q'_1 \in \texttt{rew}(Q_1, \mathcal{R})$ such that $Q'_1 \geq Q'_2$.*

Intuitively, if an operator is prunable then it guarantees that for every Q_1 more general than Q_2, the one-step rewritings of Q_2 are covered by the one-step rewritings of Q_1 or by Q_1 itself. The following lemma states that it can be generalized to k-rewritings.

Lemma 1. *Let $\texttt{rew}$ be a prunable rewriting operator, and let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two sets of queries. If $\mathcal{Q}_1 \geq \mathcal{Q}_2$, then $W_\infty(\mathcal{Q}_1, \mathcal{R}) \geq W_\infty(\mathcal{Q}_2, \mathcal{R})$.*

This lemma would not be sufficient to prove the correctness of Algorithm 1. We need a stronger version, which checks that a query whose 1-rewritings are covered needs not to be explored.

Lemma 2. *Let $\texttt{rew}$ be a prunable rewriting operator, and let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two sets of queries. If $(\mathcal{Q}_1 \cup \mathcal{Q}_2) \geq \texttt{rew}(\mathcal{Q}_1, \mathcal{R})$, then $(\mathcal{Q}_1 \cup W_\infty(\mathcal{Q}_2, \mathcal{R})) \geq W_\infty(\mathcal{Q}_1 \cup \mathcal{Q}_2, \mathcal{R})$.*

Finally, the correctness of Algorithm 1 is based on the following loop invariants.

Property 1 (Invariants of Algorithm 1). Let $\texttt{rew}$ be a prunable rewriting operator. After each iteration of the while loop of Algorithm 1, the following properties hold:

1. $\mathcal{Q}_E \subseteq \mathcal{Q}_F \subseteq W_\infty(Q, \mathcal{R})$;
2. $\mathcal{Q}_F \geq \texttt{rew}(\mathcal{Q}_F \setminus \mathcal{Q}_E, \mathcal{R})$;
3. $(\mathcal{Q}_F \cup W_\infty(\mathcal{Q}_E, \mathcal{R})) \geq W_\infty(Q, \mathcal{R})$;
4. for all distinct $Q, Q' \in \mathcal{Q}_F$, $Q \not\geq Q'$ and $Q' \not\geq Q$.

Theorem 1. *If $\texttt{rew}$ is prunable, then the output of Algorithm 1 is a minimal cover of $W_\infty(Q, \mathcal{R})$.*

3.3 Preserving Soundness and Completeness

Of course, having a prunable rewriting operator is not a sufficient condition for the soundness and completeness of the obtained rewriting set w.r.t. the usual first-order semantics. This is why we consider two further properties of a rewriting operator, namely soundness and completeness.

Definition 9 (Soundness/completeness of a rewriting operator). *Let $\texttt{rew}$ be a rewriting operator. $\texttt{rew}$ is sound if for any set of rules $\mathcal{R}$, for any query Q, for any $Q' \in \texttt{rew}(Q, \mathcal{R})$, for any fact F, $F \models Q'$ implies that $F, \mathcal{R} \models Q$. $\texttt{rew}$ is complete if for any set of rules $\mathcal{R}$, for any query Q, for any fact F s.t. $F, \mathcal{R} \models Q$, there exists $Q' \in W_\infty(Q, \mathcal{R})$ s.t. $F \models Q'$.*

Property 2. If `rew` is sound, then the output of Algorithm 1 is a sound rewriting set of Q and $\mathcal{R}$.

More surprisingly, using a complete rewriting operator in Algorithm 1 does not ensure that the output is a complete rewriting set. While this will be shown with some details in the next section, let us state that if the operator is moreover prunable, then the output set of Algorithm 1 is complete.

Property 3. If `rew` is prunable and complete, then the output of Algorithm 1 is a complete rewriting set of Q and $\mathcal{R}$.

3.4 Termination of the algorithm

We last define a condition on the rewriting operator that ensures that Algorithm 1 halts. In such case, the output needs to be finite, hence the definition of *finite coverability*.

Definition 10 (Finite coverability). *Let $\mathcal{R}$ be a set of rules. A rewriting operator `rew` is* finitely coverable *w.r.t. $\mathcal{R}$ if for every query Q there exists an integer i such that $W_i(Q, \mathcal{R}) \geq W_\infty(Q, \mathcal{R})$.*

The next property states that Algorithm 1 halts in such cases, that is, it halts each time its output is finite.

Property 4. Let $\mathcal{R}$ be a set of rules, Q be a query, and *rew* be a finitely coverable set operator w.r.t. $\mathcal{R}$. Algorithm 1 halts on $\mathcal{R}, Q,$ `rew`.

4 Piece-based Rewriting Revisited

In this section, we consider the framework of piece-unifiers. We first recall basic definitions and results. Note that we provide an alternative definition of piece-unifiers, which we will reuse in the next section to define a new rewriting operator.

4.1 Piece-based Rewriting

As detailed in [KLMT12] and shown in Example 1, existential variables in rule heads induce a structure that has to be taken into account in the rewriting step. Thus, instead of unifying the query and a rule head atom by atom, we process *subsets* of atoms.

Example 1. Let the rule $R = \forall x\ (q(x) \rightarrow \exists y\ p(x, y))$ and the Boolean CQ $Q = \exists u \exists v \exists w (p(u, v) \wedge p(w, v) \wedge r(u, w))$. Assume we want to unify the atom $p(u, v)$ from Q with $p(x, y)$, for instance by a substitution $\{(u, x), (v, y)\}$. Since v is unified with the existential variable y, all other atoms containing v must also be considered: indeed, simply rewriting Q into $Q_1 = q(x) \wedge p(w, y) \wedge r(x, w)$ would be unsound: intuitively, the fact that the atoms $p(u, v)$ and $p(w, v)$ in Q share a variable would be lost in atoms $q(x)$ and $p(w, y)$; for instance the fact $F = q(a) \wedge p(b, c) \wedge r(a, b)$ would answer Q_1 despite Q is not entailed by $(F, \{R\})$. Thus, $p(u, v)$ and $p(w, v)$ have to be both unified with the head of R, for instance by means of the following substitution: $\{(u, x), (v, y), (w, x)\}$. $\{p(u, v), p(w, v)\}$ is called a piece (as precisely defined below). The corresponding rewriting of Q is $q(x) \wedge r(x, x)$.

A piece-unifier "unifies" a subset Q' of Q with a subset H' of $\mathit{head}(R)$, in the sense that the associated substitution u is such that $u(Q') = u(H')$. Given a piece-unifier, Q is partitioned into "pieces", which are minimal subsets of atoms that must processed together. More specifically, let us call *cutpoints* the variables from Q' that are not unified with existential variables from H' (i.e., they are unified with frontier variables or constants); then a *piece* in Q is a minimal non-empty subset of atoms "glued" by variables other than cutpoints: for all atoms a and a' in Q, if a and a' share a variable that is not a cutpoint, then a and a' are in the same piece.

We call *separating variables* of Q' the variables occurring both in Q' and $Q \setminus Q'$. Condition 2 of the following piece-unifier definition (Def. 11) ensures that a separating variable is necessarily a cutpoint. It follows that Q' is composed of pieces: indeed, an existential variable from H' is necessarily unified with a non-separating variable from Q', say x, which ensures that all atoms from Q' in which x occurs are also part of Q'.

In this paper, we give a definition of piece-unifiers based on partitions instead of substitutions, which simplifies subsequent notions. To a substitution u from a set of variables E_1 to a set of terms E_2 can be assigned a *partition* P_u of $E_1 \cup E_2$ such that two terms are in the same class of P_u if and only if they are merged by u; more specifically, we consider the equivalence classes of the reflexive and transitive closure of the following relation $\sim$: $t \sim t'$ if $u(t) = t'$. Conversely, to a partition on a set of terms E, such that no class contains two constants, can be assigned a substitution obtained by selecting an element of each class with giving priority to constants. If we consider a total order on terms, such that constants are smaller than variables, then a unique substitution is obtained by taking the smallest element in each class. We call *admissible partition* a partition such that no class contains two constants.

The set of all partitions over a given set is structured in a *lattice* by the "*coarser than*" relation (given two partitions P_1 and P_2, P_1 is coarser than P_2, denoted by $P_2 \geq P_1$, if every class of P_2 is included in a class of P_1).[2] The *greatest lower bound* of two partitions is obtained by making the union of their non-disjoint classes. If we restrict our attention to admissible partitions, then two partitions may not have a greatest lower bound since the union of classes may lead to a non-admissible partition. We say that two admissible partitions are *compatible* if their greatest lower bound is an admissible partition.

In the following definition of a piece-unifier we assume that Q and R have disjoint sets of variables.

Definition 11. *[Piece-Unifier] A piece-unifier of Q with R is $\mu = (Q', H', P_u)$, where $Q' \neq \emptyset$, $Q' \subseteq Q$, $H' \subseteq \mathsf{head}(R)$ and P_u is a partition on $\mathsf{terms}(Q') \cup \mathsf{terms}(H')$ s.t.:*

- *P_u is admissible, i.e., no class in P_u contains two constants;*
- *if a class in P_u contains an existential variable (from H') then the other terms in the class can only be non-separating variables from Q'.*
- *let u be the substitution associated with P_u obtained by selecting the smallest element in each class, according to the following order: constants $<$ existential variables $<$ other variables; then $u(H') = u(Q')$.*

[2] Usually, the notation $\leq$ means "finer than". We adopt the converse convention, which is more in line with substitutions and the $\leq$ preorder on CQs.

This definition corresponds to the definition of [KLMT12], except that it considers moreover that variables from the query are necessarily substituted by variables from the rule, which would mean here that frontier variables come before variables from Q'.

Actually, not all piece-unifiers are useful: in the next sections, we will refer to most general piece-unifiers.

Definition 12. *Given two piece-unifiers defined on the same subsets of a query and a rule head, $\mu_1 = (Q', H', P_{u_1})$ and $\mu_2 = (Q', H', P_{u_2})$, we say that μ_1 is* more general than *μ_2 (notation $\mu_1 \geq \mu_2$) if P_{u_2} is coarser than P_{u_1} (i.e., $P_{u_1} \geq P_{u_2}$). A piece-unifier $\mu = (Q', H', P_u)$ is called a* most general *piece-unifier if no other piece-unifier on Q' and H' is strictly more general than μ.*

Definition 13 (One-step Rewriting). *Given a piece-unifier $\mu = (Q', H', P_u)$ of Q with R, the* one-step rewriting *of Q according to μ, denoted by $\beta(Q, R, \mu)$, is $u(\mathsf{body}(R)) \cup u(Q \setminus Q')$, where u is a substitution associated with P_u.*

Definition 14 ($\mathcal{R}$-rewriting of Q). *An* $\mathcal{R}$-rewriting of Q *is a CQ Q_k obtained by a finite sequence $(Q_0 = Q), Q_1, \ldots, Q_k$ s.t. for all $0 \leq i < k$, there is $R_i \in \mathcal{R}$ and a piece-unifier μ_i of Q_i with R_i s.t. $Q_{i+1} = \beta(Q_i, R_i, \mu_i)$.*

The next theorem states that piece-based backward chaining is logically sound and complete.

Theorem 2. *[SM96] Let a KB $\mathcal{K} = (F, \mathcal{R})$ and a Boolean CQ Q. Then $F, \mathcal{R} \models Q$ iff there is an $\mathcal{R}$-rewriting of Q that can be mapped to F.*

4.2 Piece-based Rewriting Operator

It follows from Theorem 2 that a sound and complete rewriting operator can be based on piece-unifiers: $\beta(Q, \mathcal{R})$ is the set of all one-step rewritings of Q according to a piece-unifier of Q with a rule $R \in \mathcal{R}$.

In [KLMT12] the study of piece-unifiers with specific properties is restricted to rules with a head composed of a single atom. This restriction can be done without loss of generality since any existential rule can be decomposed into an equivalent set of rules with an atomic head. It simplifies some notions and computations. In order to be able to rely on previous proofs, we assume in the sequel of this paper that rules have atomic heads. Moreover, this is in line with our current implementation of this framework. Note however that all results remain true in the general case.

Property 5. The piece-based rewriting operator is sound, complete and prunable; this property is still true if only most general piece-unifiers are considered.

Since the entailment problem is not decidable, Algorithm 1 instantiated with piece-unifiers does not halt in general. A set of rules $\mathcal{R}$ admitting a finite minimal set of $\mathcal{R}$-rewritings for any query is called a finite unification set (*fus*) [BLMS11]. Several *fus* classes of rules have been exhibited in the literature: atomic-body [BLMS09], also known as linear TGDs [CGL09], domain-restricted [BLMS09], (join-)sticky [CGP10]. Since for any finite set S of $\mathcal{R}$-rewritings, there exists an integer k such that $S \subseteq W_k(Q, \mathcal{R})$, Algorithm 1 instantiated with piece-unifiers, possibly restricted to most general piece-unifiers, halts for any fus.

Property 6. The piece-based rewriting operator is finitely coverable for any finite unification set of rules; this is still true if only most general piece-unifiers are considered.

4.3 Single-Piece-based Rewriting is not Prunable

A piece-unifer $\mu = (Q', H', P_u)$ is said to be a *single-piece* unifier if Q' is a single piece. In [KLMT12] (Theorems 4 and 5) it is shown that (most general) single piece-unifiers provide a complete operator. However, the restriction to single-piece unifiers is not compatible with selecting most general rewritings at each step, as done in Algorithm 1. We study below some examples that illustrate this incompatibility (we omit H' in these examples since all rule heads are atomic).

Example 2 (Basic example). Let $Q = p(y, z) \wedge p(z, y)$ and $R = r(x, x) \rightarrow p(x, x)$. There are two single-piece unifiers of Q with R, $\mu_1 = (p(y, z), p(x, x), \{\{x, y, z\}\})$ and $\mu_2 = (p(z, y), p(x, x), \{\{x, y, z\}\})$, which yield the same rewriting, e.g. $Q_1 = r(x, x) \wedge p(x, x)$. There is also a two-piece unifier $\mu = (Q, p(x, x), \{\{x, y, z\}\})$, which yields e.g. $Q' = r(x, x)$. A query equivalent to Q' can be obtained from Q_1 by a further single-piece unification. Now, assume that we restrict unifiers to single-piece unifiers *and* keep most general rewritings at each step. Since $Q \geq Q_1$, Q_1 is not kept, so Q' will never be generated, whereas it is incomparable with Q.

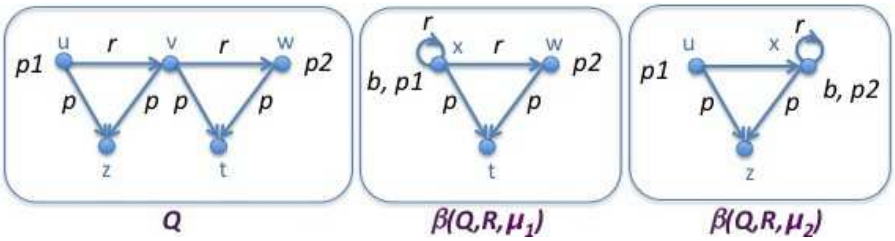

Fig. 1. The queries in Example 3

Example 3. This example has two interesting characteristics: (1) it uses unary/binary predicates only (2) it uses a very simple rule expressible with any lightweight description logic, i.e., a linear existential rule where no variable appears twice in the head or the body. Let $Q = r(u, v) \wedge r(v, w) \wedge p(u, z) \wedge p(v, z) \wedge p(v, t) \wedge p(w, t) \wedge p_1(u) \wedge p_2(w)$ (see Figure 1) and $R = b(x) \rightarrow p(x, y)$. Note that Q is not redundant. There are two single-piece unifiers of Q with R, say μ_1 and μ_2, with pieces $P_1 = \{p(u, z), p(v, z)\}$ and $P_2 = \{p(v, t), p(w, t)\}$ respectively. The obtained queries are pictured in Figure 1. These queries are both more specific than Q. The removal would prevent the generation of a query equivalent to $r(x, x), p_1(x), p_2(x), b(x)$, which could be generated from Q with a two-piece unifier.

Property 7. The single-piece-based operator is sound, complete, finitely coverable for any finite unification set of rules, but it is not prunable.

However, single-piece unifiers can still be used as an algorithmic brick to compute general piece-unifiers. The obvious way of doing consists of merging "compatible" single-piece unifiers to compute all piece-unifiers. In the next section, we present another method: we aggregate single-piece unifiers in order to obtain a notion more general than a piece-unifier, that we call an aggregated unifier.

5 Aggregated Piece-based Rewriting

In this section we define a way of combining most general single-piece unifiers that allows to retrieve the desired prunability property. Two versions of rewriting operators based on this combination are proposed. The first one, called single-rule aggregator, consists in gathering "compatible" sequences of most general single-piece unifiers of a query Q with the *same* rule R into a single unifier. The second one, called the all-rule aggregator, is an extension of the first one that gathers compatible sequences of most general single-piece unifiers of Q with possibly *different* rules into a single unifier.

For the following definitions, we consider partitions of possibly distinct subsets. Alternatively, given partitions P_1 of set S_1 and P_2 of set S_2, we can extend them to partitions on the same set $S_1 \cup S_2$ by adding each missing element in its own class.

When we combine two piece-unifiers relative to the same rule, the variables of the rule are renamed. Thus in the following, $R_1 \dots R_k$ denote distinct copies of possibly less than k distinct rules.

Definition 15 (Compatible Piece-Unifiers). *Let $\mathcal{U} = \{\mu_1 = (Q'_1, H'_1, P_1) \dots \mu_k = (Q'_k, H'_k, P_k)\}$ a set of piece-unifiers of Q with rules $R_1 \dots R_k$ (respectively). $\mathcal{U}$ is said to be* compatible *if (1) all Q'_i and Q'_j are pairwise disjoint; (2) the greatest lower bound of $P_1 \dots P_k$ is admissible.*

Note that the following additional condition will be always fulfilled for compatible piece-unifiers: for all i and j, the sets of variables of H'_i and H'_j are pairwise disjoint.

Definition 16 (Aggregated unifier). *Let $\mathcal{U} = \{\mu_1 = (Q'_1, H'_1, P_1), \dots, \mu_k = (Q'_k, H'_k, P_k)\}$ be a compatible set of piece-unifiers of Q with rules $R_1 \dots R_k$. An* aggregated unifier *of Q with $R_1 \dots R_k$ w.r.t. $\mathcal{U}$ is $\mu = (Q', H', P)$ where: (1) $Q' = Q'_1 \cup \dots \cup Q'_k$; (2) $H' = H'_1 \cup \dots \cup H'_k$; (3) P is the greatest lower bound of $P_1 \dots P_k$. It is said to be* single-piece *if all the piece-unifiers of $\mathcal{U}$ are single-piece. It is said to be* most general *if all the piece-unifiers of $\mathcal{U}$ are most general.*

Definition 17 (Aggregation). *The aggregation of a set of rules $\mathcal{R} = \{R_1 \dots R_k\}$ is the rule $R = \mathsf{body}(R_1) \wedge \dots \wedge \mathsf{body}(R_k) \rightarrow \mathsf{head}(R_1) \wedge \dots \wedge \mathsf{head}(R_k)$ (where we assume that all rules have disjoint sets of variables).*

Property 8. Let Q be a query and $\mathcal{U} = \{\mu_1 = (Q'_1, H'_1, P_1) \dots \mu_k = (Q'_k, H'_k, P_k)\}$ be a compatible set of piece-unifiers of Q with $R_1 \dots R_k$. Then the aggregated unifier of $\mathcal{U}$ is a piece-unifier of Q with the aggregation of $\{R_1 \dots R_k\}$.

We call single-rule aggregator (resp. all-rule aggregator) and denote by `sra` (resp. `ara`) the rewriting operator that assigns to a query Q and a set of rules $\mathcal{R}$, the set of

all the queries Q_i such that Q_i is the one-step rewriting of Q with an aggregated unifier that aggregates most general compatible single-piece unifiers of Q with the *same* rule (resp. with *any* rule) of $\mathcal{R}$.

Property 9. `sra` and `ara` are sound, complete, prunable, and finitely coverable for any finite unification set of rules.

6 Optimization, Experiments and Perspectives

In this section, we present some ongoing work: an optimization that makes `ara` more efficient than `sra`, as well as experiments. We conclude with further work.

6.1 Optimization

Operators `sra` and `ara` generate several times the same rewriting with sequences of aggregated unifiers that only differ with respect to the order in which parts of the query are unified. The situation is even worse for `ara`, as illustrated by the following example and Figure 2.

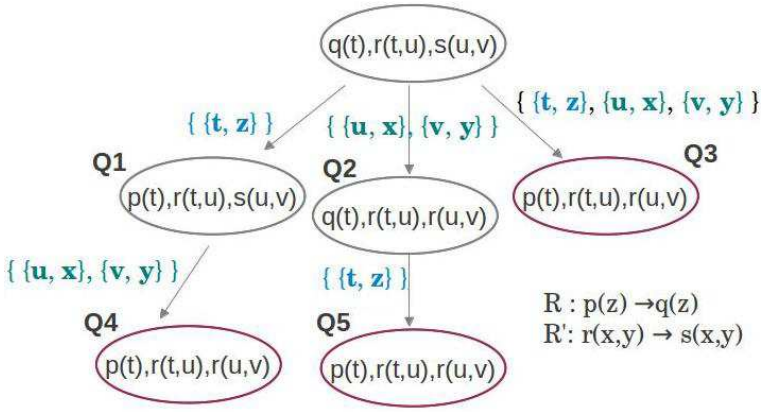

Fig. 2. The same rewritings are generated several times

Example 4. Let $\mathcal{R} = \{p(z) \rightarrow q(z), r(x,y) \rightarrow s(x,y)\}$ and $Q = q(t) \wedge r(t,u) \wedge s(u,v)$. There are two most general single-piece unifiers of Q with a rule of $\mathcal{R}$. $\mu_1 = (\{q(t)\}, \{q(z)\}, \{\{t,z\}\})$ and $\mu_2 = (\{s(u,v)\}, \{s(x,y)\}, \{\{u,x\}, \{v,y\}\})$. From these compatible single-piece unifiers `ara` will compute three aggregated unifiers: the first ones are identical to μ_1 and μ_2 and the last one is $\mu_3 = (\{q(t), s(u,v)\}, \{s(x,y), q(z)\}, \{\{t,z\}\{u,x\}\{v,y\}\})$, which can be seen as a piece-unifier of Q with the aggregated rule $r(x,y) \wedge p(z) \rightarrow s(x,y) \wedge q(z)$. The rewritings produced by μ_1, μ_2 and μ_3 are respectively $Q_1 = p(t) \wedge r(t,u) \wedge s(u,v)$,

$Q_2 = q(t) \wedge r(t,u) \wedge r(u,v)$ and $Q_3 = p(t) \wedge r(t,u) \wedge r(u,v)$. Note that `sra` will not produce Q_3. At the next rewriting step, a piece-unifier identical to μ_1 will be applicable to Q_2 and will produce Q_5 equivalent to Q_3. Symmetrically, a piece-unifier identical to μ_2 will be applicable to Q_1 and will produce Q_4 equivalent to Q_3 (see Figure 2).

A simple way of avoiding these equivalent rewritings in `ara` is as follows: we mark the newly added atoms when a rewriting is generated and consider only unifications involving at least a marked atom. Indeed, unifications involving only non-marked atoms have already been performed at a former step. More specifically, we will compute aggregated unifiers only on the compatible sets of single-rule aggregated unifiers that unify at least one marked atom i.e., an atom added at the previous rewriting step.

Example 4 (continued). We come back to the previous example by marking (i.e., underlying) the atoms just added. The rewriting produced by μ_1, μ_2 and μ_3 are respectively $Q_1 = \underline{p(t)} \wedge r(t,u) \wedge s(u,v)$, $Q_2 = q(t) \wedge r(t,u) \wedge \underline{r(u,v)}$ and $Q_3 = \underline{p(t)} \wedge r(t,u) \wedge \underline{r(u,v)}$. At the next step, there is no unifier of Q_1 and Q_2 with a rule of $\mathcal{R}$ using a marked atom. Indeed, there is a unifier of Q_2 with the first rule of $\mathcal{R}$ identical to μ_1 but it does not use a marked atom. Symmetrically, the unifier of Q_1 with the second rule of $\mathcal{R}$ identical to μ_2 does not use a marked atom. Thus Q_4 and Q_5 will not be produced.

This optimization keeps soundness and completeness of the operator but may be not its prunability. We have checked in all our experiments that the same rewriting set is finally output with and without this optimization. As developed in the next section, experiments also show that this operator is more efficient than the previous ones, in the sense that it generates significantly less queries. We thus have a candidate sound and complete rewriting operator, faster and that practically outputs a sound and complete rewriting set. Note that, in case it would not have the desired theoretical properties, this operator would still be interesting in applicative settings where efficiency matters more than a theoretical guarantee of completeness, since the difference could not be detected experimentally.

6.2 Experiments

The generic breadth-first algorithm has been implemented in Java and instantiated with the different rewriting operators, namely the single-rule aggregator, the all-rule aggregator without optimization, and the all-rule aggregator with optimization.

First experiments were led on sets of existential rules obtained by translation from ontologies expressed in DL-Lite$_\mathcal{R}$ developed in several research projects, namely ADOLENA (A), STOCKEXCHANGE (S), UNIVERSITY (U) and VICODI (V). See [GOP11] for more details. The obtained rules have atomic head and body, which corresponds to the linear Datalog+/- fragment. The associated queries have been generated by Sygenia [ISG12]. Sygenia provided us with 114, 185, 81 and 102 queries for ontologies A, S, U and V respectively.

Table 1 presents the sum of the number of generated CQ rewritings (# generated) for each ontology and each of the operators (`sra` : single-rule aggregator, `ara` : all-rule aggregator, `ara-opt` : all-rule aggregator with optimization). The generated rewritings

are all the rewritings built during the rewriting process (excluding the initial query and possibly including some multi-occurrences of the same rewritings). We also mention the sum of the cardinalities of the final output sets (# output), which is the same for all operators. The all-rule aggregator without optimization is always worse than the single-rule aggregator, since by definition it generates a superset of `sra` rewritings; however its optimized version is significantly better than `sra`, especially for ontology A. We believe that the difference between both operators should increase with the complexity of the ontologies and the queries. However, complex real-world ontologies and queries are lacking for now.

rule base	# output	`sra` : # generated	`ara` : # generated	`ara-opt` : # generated
A	3209	146 523	357 584	62 813
S	557	6515	13246	6143
U	486	2122	3484	2201
V	2694	5318	7522	3286

Table 1. Results with `sra`, `ara` and `ara-opt`

6.3 Perspectives

As explained above, the optimized all-rule aggregator can be seen as an interesting candidate operator, which is sound and complete, practically prunable, and more efficient than the classical piece-based operator. However, prunability and efficiency have still to be studied from a theoretical viewpoint. Further work includes implementing other optimisations, by exploiting for instance dependencies between rules to select the rules to be considered at each step, extending algorithms to rules with non-atomic head, combining aggregation with query factorization techniques, such as those developed in [Tho13], as well as experimenting the algorithms on more complex queries and ontologies.

Acknowledgements. This work was partially funded by the ANR project PAGODA (ANR-12-JS02-007-01).

References

BLMS09. J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending decidable cases for rules with existential variables. In *IJCAI'09*, pages 677–682, 2009.

BLMS11. J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9-10):1620–1654, 2011.

CGK08. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR'08*, pages 70–80, 2008.

CGL$^+$07. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

CGL09. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS'09*, pages 77–86, 2009.

CGP10. A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in datalog+/-. In *RR'10*, pages 1–17, 2010.

GOP11. G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE'11*, pages 2–13, 2011.

ISG12. Martha Imprialou, Giorgos Stoilos, and Bernardo Cuenca Grau. Benchmarking ontology-based query rewriting systems. In *AAAI*, 2012.

KLMT12. M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. A sound and complete backward chaining algorithm for existential rules. In M. Krötzsch and U. Straccia, editors, *RR*, volume 7497 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 2012.

KLMT13. M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. On the Exploration of the Query Rewriting Space with Existential Rules. Technical Report RR-13016, LIRMM, GraphIK - INRIA Sophia Antipolis, April 2013.

KLT$^+$11. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The Combined Approach to Ontology-Based Data Access. In *IJCAI*, pages 2656–2661, 2011.

KR11. M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI'11*, pages 963–968, 2011.

LTW09. C. Lutz, D. Toman, and F. Wolter. Conjunctive query answering in the description logic el using a relational database system. In *IJCAI'09*, pages 2070–2075, 2009.

PUHM09. H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient query answering for owl 2. In *ISWC'09*, pages 489–504, 2009.

RA10. R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *KR'10*, 2010.

RMC12. M. Rodriguez-Muro and D. Calvanese. High performance query answering over DL-lite ontologies. In *KR*, 2012.

SM96. E. Salvat and M.-L. Mugnier. Sound and Complete Forward and Backward Chainings of Graph Rules. In *ICCS'96*, volume 1115 of *LNAI*, pages 248–262. Springer, 1996.

Tho13. M. Thomazo. Compact rewriting for existential rules. In *IJCAI*, 2013.

VSS12. T. Venetis, G. Stoilos, and G. B. Stamou. Incremental query rewriting for OWL 2 QL. In *Description Logics*, 2012.